Project Proposal

10th August 2019

Simon Curtis
44318923
s4431892@student.uq.edu.au

Investigation of high-performance packet capture using commodity hardware.

Abstract:

Packet capture at bandwidth speeds of 10Gbps+ is not viable using commodity hardware and Linux kernel without packet loss. This is due to the high overhead of the Linux networking stack. Several frameworks are available that bypass the Linux networking stack and deliver packets directly to userspace for processing, and can improve performance. These are: DPDK, PF_RING_ZC, NETMAP and AF_PACKET. This report proposes a project where these frameworks would be tested in their ability to capture packets passing through an interface, and export them as NetFlow.

1.0 Introduction

It is now commonplace for networks to reach speeds of 10 or 40Gbps in universities, businesses and home environments. High-end network interface cards (NICS) can even reach speeds of 200Gbps [1]. These speeds are becoming so fast that it is hard for commodity hardware and software to keep up with network capabilities, and they are constantly becoming the bottlenecks in high-speed network operations.

Things such as packet capture, analysis, forwarding and firewalls all require special hardware to operate at these speeds without latency or packet loss. This can be expensive [2] and hard to set up, but using commodity hardware and a default Linux network stack often does not provide the required performance. This is because the kernel networking stack is slow, with multiple steps to pass packets up to userspace, packet data being copied multiple times and costly context switching.

One technology that provides a workaround to this is kernel bypass technology. It uses a variety of techniques to speed up the processing of packets by bypassing the kernel and allowing a user process to access the packets directly from the NIC.

In recent years, several free, open-source software frameworks have emerged that provide a variety of different services to improve network functionality on commodity hardware, including packet processing. Some of the more promising include: DPDK, PF_RING_ZC, NETMAP and AF_PACKET. While these frameworks all differ slightly in implementation, purpose and features offered, they all provide some speedup in packet processing.

This report proposes a method to test and compare the packet processing ability of the frameworks mentioned above, whilst implementing a useful network monitoring feature (NetFlow) by creating a common NetFlow exporter that can interface easily to the different frameworks in a modular manner.

2.0 Topic Definition

2.1 Purpose

The purpose of this thesis is to perform an investigation into various software frameworks (DPDK, PF_RING_ZC, NETMAP and AF_PACKET) that provide high-performance packet capture on commodity hardware. These software frameworks will be tested and compared based on their ability to perform packet capture and exportation via NetFlow. Improvements and optimisations will be applied to attempt to further advance performance.

2.2 Aims

- To compare the packet capture performance of various available cutting edge technologies at a high bandwidth on commodity hardware.
- To test the limits of fast packet capture for commodity hardware running the Linux operating system.
- To implement a NetFlow exporter which easily interfaces to the different technologies.
- To be able to capture packets and export as NetFlow at a bandwidth of 10Gb/s.
- To investigate the various techniques that are at the cutting edge of fast packet capture

2.3 Coverage

In scope:

- Packet capture
- NetFlow exporting
- dpdk, pr_ring_zc, af_packet, netmap
- Kernel bypass & high speed technologies
- Working with Linux OS

Out of scope:

- Packet forwarding
- Deep packet inspection / analysis
- NetFlow collection
- Creating a low-level packet capture framework
- Using specialised hardware
- PACKET_NMAP, SNABB SWITCH, PACKET_SHADER and PFQ
- Modifying kernel stack

2.4 Relevance

- 200gbps+ Ethernet exists today and is getting faster. Software can’t keep up.
- Performance hardware is expensive [2]
- Linux is a popular and practical OS in today's computing world.
- These frameworks are the cutting edge of fast packet capture [3]

3.0 Background Review

3.1 Information, Background Material & Required Knowledge

The current networking stack for a standard Linux distribution can be split into 3 sections: the physical layer, user space and kernel space. The kernel space can be split into a further 5 sub-layers: System call Interface, Protocol Agnostic Interface, Network Protocol layer, Device Agnostic Interface and Device Drivers. [4]

Each of these layers plays a role in moving a packet from the application process to the network device, and vice versa. The system call interface allows a user process to access a socket via common read() and write() operations on a file descriptor. This is the starting point of the network stack. The protocol-agnostic interface calls the sendmsg() function pointed to by the ops field of the socket struct. For example, in the case of the INET socket, it would call inet_sendmesg(). The inet_sendmesg() function will then call a protocol-specific function. For example, if the protocol used is TCP, then the kernel will now enter the TCP handling code, which breaks the packet up into chunks according to the TCP protocol and writes them to sk_buff structures. The sk_buff structures then have IP headers applied and are passed through various firewall checks and/or NAT/masquerading. The sk_buff struct is now passed to the device-agnostic layer, where Qos and queueing routines are applied. The last step happens when the packet is dequeued and then the buffer is passed to the device driver when ready, for sending out the interface. [5]

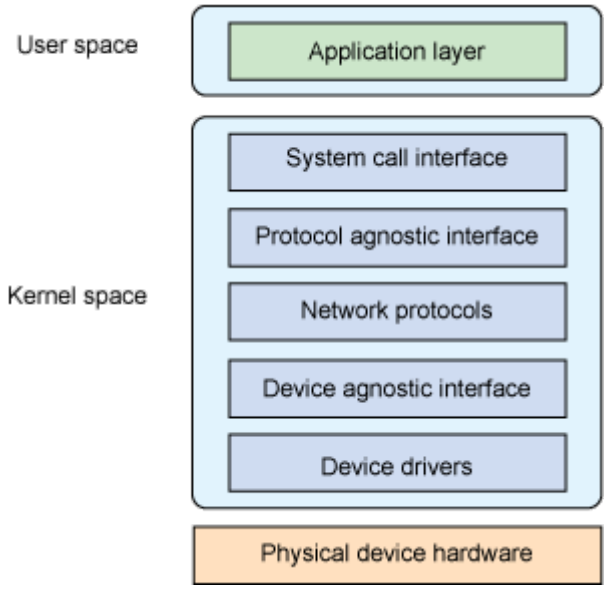

Figure 1: Linux Networking Stack [6]

The kernel, through the use of this stack, provides a variety of services to applications wishing to utilise the network. Things such as checksum handling, routing, transport, network, link-layer protocols, interfacing to hardware and useful networking APIs such as sockets, are all offered to the application by the kernel. This make network programming very simple for application processes. However, using the default kernel networking stack has several limitations. For one, the main data structures used in the kernel for handling devices and packets (e.g. the sk_buff structure which holds packet data) are constantly copied to different locations and structures [7]. This can be quite costly when done at high rates. Another downside is the fact that there needs to be a context switch from user mode to kernel mode. This wastes considerable CPU cycles when done frequently.

Kernel Bypass is the process of circumventing the kernel networking stack. This is usually achieved via a software framework that reads the packets directly from the interface and then passes them to applications running on top of them.

Several packet processing acceleration techniques are commonly used by software frameworks performing kernel bypass. Some of these include:

- Polling for received packets instead of relying on interrupts.
- Preallocating packet buffers.
- Direct Memory Access of packet data to user memory to avoid copying between the kernel and user memory.
- Batch processing multiple packets at a time to avoid excessive API calls.
- RSS (Receive Side Scaling) or multi-queue support. This effectively allows multicore support by queueing packets for each CPU available.
- Some software frameworks have explored the idea of using GPUs for packet processing, as they might be able to analyse multiple packets at once because of their highly parallel computing power.

[3]

Packet capture is the process of collecting the raw packets straight off an interface. Usually for inspection and analysis or forwarding.

NetFlow is a service provided to network administrators to access IP flow information from their data networks [8]. A flow is defined as: “a unidirectional sequence of packets with some common properties that pass through a network device.”. [8]
Various versions of Netflow exists (v5, v9 IPFIX) but they are all based on the same basic principals. NetFlow setups exist as an exporter and a collector. The Netflow exporter usually runs on routers and switches and collects flow information as data passes through the interfaces. The exporter then sends flow records to a collector, which stores the flow records for later analysis. Flow records contain a collection of useful fields, such as packets that passed through an interface. Analysing NetFlow can help identify network performance issues, as well as identify any suspicious or unusual network activity.

3.2 Overview of technologies

AF_PACKET:

AF_PACKET is a type of address family that can be used in Linux to send and receive packets at the data link, or physical layer [9]. It is used by passing the value AF_PACKET to the socket call if the application has CAP_NET_RAW capabilities. AF_PACKET reduces the overhead of using most of the Linux stack, and can be configured along with other options such as PACKET_RX_RING to use a ring buffer like in dpdk or PACKET_FANOUT to scale processing across threads.

PF_RING_ZC:

PF_RING_ZC is a kernel module built by Ntop for the Linux operating system [10]. It allows a high rate of packet processing and supposedly offers line-rate performance into the 10Gb/s [11]. It is a mostly free software although some features are paid for. The zero-copy API is application and developer focussed, rather than hardware focussed and is easier to use than some of the other frameworks. Its network driver has a feature where

you can switch between kernel bypass mode, and then back to the network stack when required.

DPDK:

DPDK (data plane development kit) is a collection of libraries to accelerate packet processing on a variety of CPU architectures [12]. It is an open-source software platform, developed by Intel for enabling fast packet I/O [13]. DPDK supports most CPU architectures and NICS. It works by using a ring buffer, similar to FreeBSD's buffring.c library. It is based on a producer-consumer model, where incoming packets are placed onto the buffer, and the application checks regularly for new packets [14]. DPDK also has many features and libraries available to make packet processing easier, such as longest prefix matching. DPDK is a highly configurable framework, capable of making the most of the supplied hardware. For a full list of features available see [15].

NETMAP:

NETMAP, is a system to give user space applications very fast access to network packets, both for receiving and transmitting [16]. However, unlike the previous frameworks, netmap still uses a partial operating system network stack for low-level operations that are potentially dangerous such as driving the NIC and validating memory. Efficiency is still provided by features such as: lightweight metadata representation, linear preallocated packet buffers, direct, protected access to packet buffers by applications, and supporting common hardware features, such as multiplexing hardware queues. Netmap attempts to make its API simple, and abstracts away hardware and specific details.

Others:

There are a variety of other software frameworks available for efficient network operations. These will be explored if time permits, however, priority will be given to the ones mentioned above. Other software frameworks that could be considered include: PACKET_NMAP, SNABB SWITCH, PACKET_SHADER and PFQ.

Comparison:

Each framework has different pros, cons and use cases. From the initial research gathered, it looks like af_packet and netmap are the easiest to use, while pf_ring_zc and dpdk are slightly harder to setup, but with more configuration and optimisation available.

3.3 Previous work

Several applications have been built upon the software frameworks in section 3. These include:

- An open-source, NetFlow v5 DPDK exporter [17]
- Ntop, a group of high-speed network applications that have been built on top of PF_RING_ZC [11]
- IP Firewall (ipfw) is a kernel-based firewall that is built on top of NETMAP and available by default on the FreeBSD operating system[18].
- DPDK open vswitch [19]

Previous research reports exist showing comparisons of the different frameworks. Although there have not been any found that compare all 4 of the above at once, or with regards to comparing them by their ability to export NetFlow, they do provide insight into the performance expected from the frameworks, and an overview of the features they offer. The results of these reports can serve as a comparison for the results of this thesis.

Comparison of Frameworks for High-Performance Packet IO [3]

Compares DPDK, NETMAP and PF_RING_ZC on their ability to forward packets. This paper also gives insight into the various techniques that fast packet capture frameworks employ to achieve performance. The results of this paper show that PF_RING_ZC and DPDK to be significantly superior to NETMAP in both throughput and latency (with DPDK slightly ahead of PF_RING_ZC). However, the report states that NETMAP has the added advantage of “interface continuity and system robustness”.

Comparing Ring-buffer–based Packet capture solutions [20]

Compares lost packets at speed between NETMAP, PF_RING_ZC and vanilla Linux. The report also compares NETMAP on BSD vs Linux. The results show that the default kernel stack was dropping 34.68% of packets, PF_RING_ZC 0.0410% and NETMAP 0.1916%. Interestingly, NATMAP had better performance when running on FreeBSD.

Comparison of High-Performance Packet Processing Frameworks on NUMA [21]

Compares the multi-queue abilities of PF_RING_ZC and DPDK using the NUMA interface. The results show that the pipeline performance of PF_RING_ZC is faster than DPDK. However, when inter-thread communication is used, DPDK is faster.

Fast User Space Packet Processing [22]

Uses NETMAP and DPDK to produce a faster version of click (a software IP router).

4.0 Methodology

The plan and setup to test the performance of the frameworks is as follows:

A test machine will be setup with the software framework installed that is to be tested (e.g. DPDK). A NetFlow exporter, interfacing to the framework should also be installed on the test machine. A high-speed traffic generator will be set up, capable of producing Gbps traffic. This traffic generator will send traffic into the interface on the test machine that is being read by the tested software framework. The test machine will then export NetFlow flow records to the NetFlow collector and the traffic may continue to its destination, for further statistics gathering. The results of the test could be analysed by viewing the Netflow data and/or viewing test machine or destination machine statistics.

Note, this setup is still open to different approaches, as there are several details and variables that still need to be explored and tested for the best approach (see below).

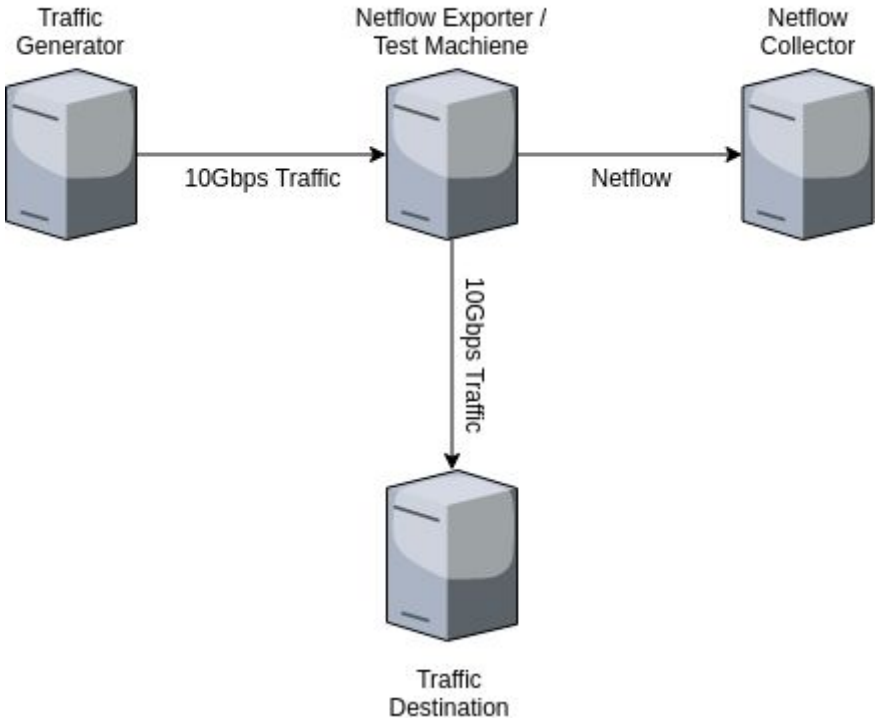

Figure 2: Setup configuration

Below are some open-ended questions and variables to be considered for the setup, along with considerations and options for each variable . At the present time, these options still need to be explored further and tested to find out what works best.

- How will traffic be generated?
    - Options for generating traffic will need to be explored and see what works best. Some possibilities include:
    - ZSEND
    - Ostinato
    - Extended ping packets
    - PacketSender
- Should there be a destination process that receives the traffic?

    - This would mean the test could gather more measurements / stats, but also means an extra machine that needs to be used. Also, the experiment is just to do packet capturing, not forwarding, so if traffic is forwarded to a destination after processing, this is technically out of scope, and may interfere with the performance of the testing.
- What should the NetFlow collector process be?
    - AKIPS
    - Solarwinds NetFlow analyser
    - nProbe
    - Many others
- Should machines be physical or virtual?
    - Virtual machines might be easier to set up as less physical computers are needed, however doing this might affect performance. It is not known if the software would work correctly / the same on a virtual machine as a physical one. So it is probably best to use physical machines. The exception is the NetFlow collector, as this shouldn't be under heavy load.
- Should there be timestamps on the packets?
    - This would mean that delay could be measured. However, something would have to be setup on the sending and receiving interfaces to do this.
- How would performance be measured?
    - Performance metrics logged on exporter machine (CPU util, mem used etc)
    - Timestamps
    - Latency/delay timing on receiving process
    - By viewing NetFlow statistics

5.0 Project Plan

5.1 Plan

- Initial research and information gathering.
- Academic integrity tutorial.
- Project proposal report.
- Setup “hello world” implementation running on a personal pc. This includes a dedicated NetFlow collector to test the exporter, the exporter itself, which will run on the ethernet card on a personal pc, a traffic generator source, and a possible traffic destination to test if the received traffic is correct.
    - NetFlow collector
    - NetFlow exporter
    - Setup traffic generator
- Measure the maximum performance possible that is reached by the default Linux kernel to see what the lower bound should be, and to provide context and comparison for the other measurements.
- Test “hello world” implementation with DPDK.
- Once this is setup, the NetFlow exporter module should be made to easily interface to the different technologies, to allow reuse of code for the exporter, and a common comparison point for the different technologies.
    - DPDK
    - PF_RING_ZC
    - NETMAP
    - AF_PACKET
- Testing of these initial implementations.
- Progress Seminar.
- After the initial implementation has basic functionality working, with low bandwidth, it would then be ideal to run the software on faster and more dedicated hardware, and begins to push the boundaries of what can be achieved with these technologies.
    - Get fast network setup
- Performance testing and analysis will then be performed on the various different technologies to compare them, and see which one is the best for the purpose of packet capture.
    - Once the highest performing framework has been selected, work will begin on how to improve it. Some ideas to improve include:
    - different NetFlow versions (v5, v9, IPFIX)
    - utilising multiple cores
    - RSS and multiple hardware queues
    - utilise GPU
    - batch processing
    - exporting via HTTP
- Poster and demonstration.
- Analyse results.
- Write up the final report.

5.2 Gantt Chart

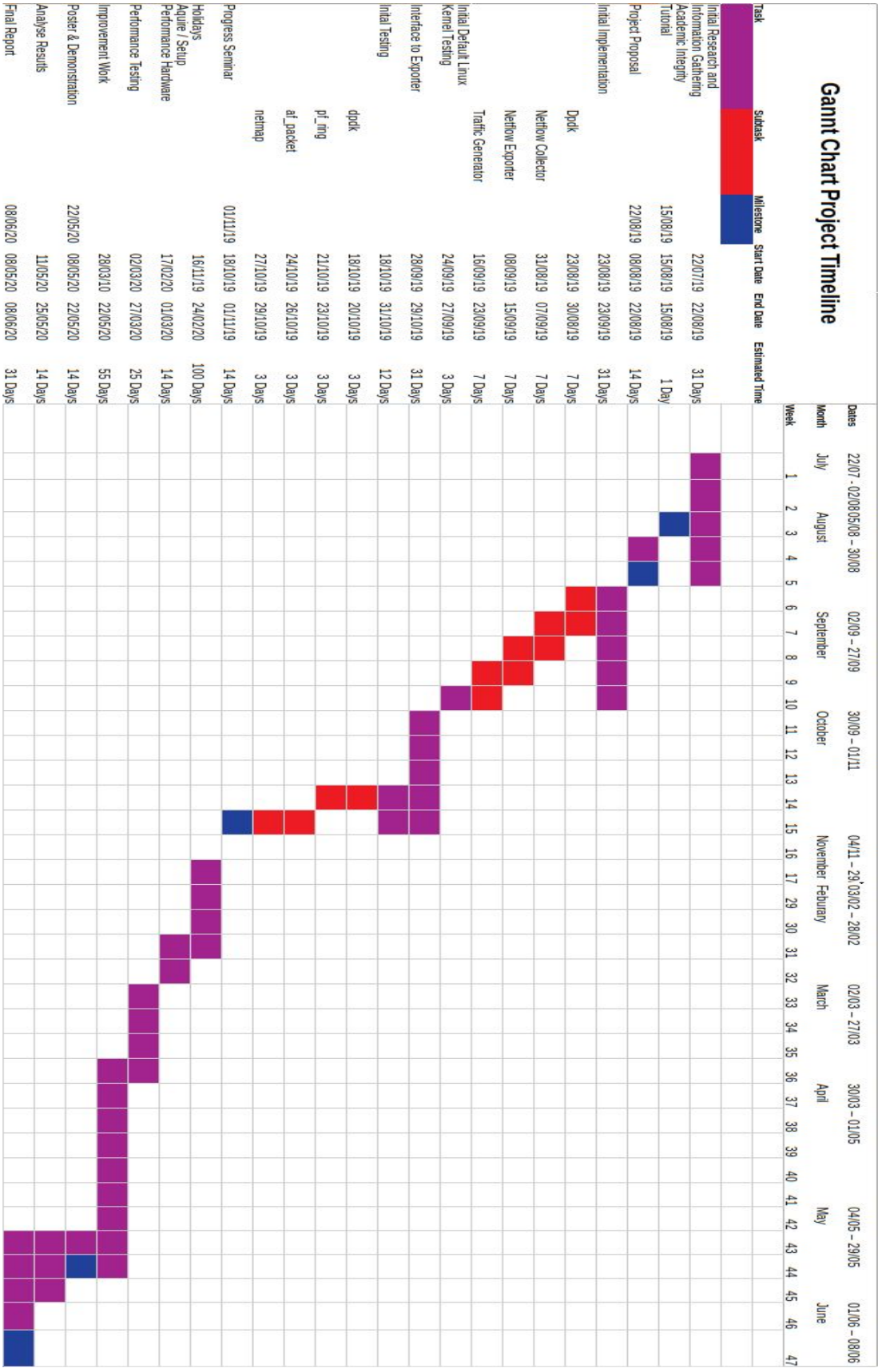

Figure 3: Project Timeline

6.0 Risk Assessment

This project is a low-risk environment. Work in the universities laboratories will be covered by the general OHS laboratory rules. The below table identifies risk in relation to project/schedule and infrastructure.

The levels of impact are defined as follows:
LOW - <= 1 week delay to project and/or insignificant disruption to infrastructure
MED - 2 - 3 weeks delay to project and/or minor or temporary disruption to infrastructure
HIGH - >4 weeks delay to project and/or major or permanent disruption to infrastructure

Name	Description	Impact	Mitigation
File Protection	Files can become lost or corrupted due to failed hardware or user error. Could lose large amounts of work and progress.	HIGH	Backup files regularly using some form of version control (e.g. git or svn).
The project falls behind schedule	If progress is slower than anticipated then there is a risk that the full project may not be completed by the deadline. (e.g. The planned steps in section 5.0 take longer than anticipated to complete.)	MED	Re-evaluate progress regularly to ensure that the project is on track for completion. If found to be behind schedule, it may be better to redefine the scope of the project. (e.g. remove some of the frameworks to be evaluated)
Unable to acquire hardware and resources	Generating and testing high-performance packet capture and exportation requires specific hardware such as high-performance network interface cards and packet generators. If this equipment is not acquired on time, it may delay the project.	LOW	Make sure specific hardware and resources are requested in advance.
Network issues caused by project	Since this project involves using high volume network traffic, it is important to ensure that this traffic is handled correctly, and does not affect the quality of service that is provided on the network. (e.g. flooding the network with packets)	MED	Test with low volumes of network traffic first, and then increase traffic once program correctness is assured. Test on isolated networks first, or, if this is not possible, non-critical networks.

Bibliography

[1]P. Enberg, "On Kernel-Bypass Networking and Programmable Packet Processing", Medium, 2019. [Online]. Available: https://medium.com/@penberg/on-kernel-bypass-networking-and-programmable-packet-processing-799609b06898. [Accessed: 14- Aug- 2019].

[2]"100% accurate, High speed packet capture solutions for Enterprises - Endace", *Endace.com*, 2019. [Online]. Available: https://www.endace.com/endace-high-speed-packet-capture-solutions/enterprise/. [Accessed: 12- Aug- 2019].

[3]S. Gallenmüller, P. Emmerich, F. Wohlfart, D. Raumer and G. Carle, "Comparison of Frameworks for High-Performance Packet IO", Technische Universität München, München, 2019.

[4]J. Buse, "Linux Network Stack", Linux.org, 2019. [Online]. Available: https://www.linux.org/threads/linux-network-stack.9065/. [Accessed: 22- Aug- 2019].

[5]S. Seth and M. Venkatesulu, TCP/IP architecture, design and implementation in Linux. Hoboken, N.J.: Wiley, 2008.

[6]T. Jones, "Anatomy of the Linux networking stack", 140.120.7.21, 2019. [Online]. Available: http://140.120.7.21/LinuxRef/Network/LinuxNetworkStack.html. [Accessed: 22- Aug- 2019].

[7]R. Rosen, Linux Kernel Networking. [New York, N.Y.]: Apress, 2014.

[8]E. B. Claise, "Cisco Systems NetFlow Services Export Version 9", IETF, 2019.

[9]packet - packet interface on device level. http://man7.org/linux/man-pages/man7/packet.7.html: Free Software Foundation, 2019.

10]"ntop/PF_RING", GitHub, 2019. [Online]. Available: https://github.com/ntop/PF_RING. [Accessed: 17- Aug- 2019].

[11]"Introducing PF_RING ZC (Zero Copy)", ntop, 2019. [Online]. Available: https://www.ntop.org/pf_ring/introducing-pf_ring-zc-zero-copy/. [Accessed: 22- Aug- 2019].

[12]"About - DPDK", DPDK, 2019. [Online]. Available: https://www.dpdk.org/about/. [Accessed: 13- Aug- 2019].

[13]"Charter - DPDK", DPDK, 2019. [Online]. Available: https://www.dpdk.org/charter. [Accessed: 17- Aug- 2019].

[14]A. Yemelianov and A. Yemelianov, "Introduction to DPDK: Architecture and Principles - Selectel Blog", *Selectel Blog*, 2019. [Online]. Available: https://blog.selectel.com/introduction-dpdk-architecture-principles/. [Accessed: 20- Aug- 2019].

[15]"2. Features Overview — Data Plane Development Kit 19.08.0 documentation", Doc.dpdk.org, 2019. [Online]. Available: https://doc.dpdk.org/guides/nics/features.html#flow-control. [Accessed: 18- Aug- 2019].

[16]L. Rizzo, "netmap: a novel framework for fast packet I/O", Universita di Pisa, 2019.

[17]*Netflow engine with DPDK support*. https://code.google.com/archive/p/netflow-dpdk/: Google, 2019.

[18]"30.4.�IPFW", *Freebsd.org*, 2019. [Online]. Available: https://www.freebsd.org/doc/handbook/firewalls-ipfw.html. [Accessed: 14- Aug- 2019].

[19]R. Giller, "Open vSwitch* with DPDK Overview", *Software.intel.com*, 2019. [Online]. Available: https://software.intel.com/en-us/articles/open-vswitch-with-dpdk-overview. [Accessed: 15- Aug- 2019].

[20]Sand, "Comparing Ring-buffer–based Packet capture solutions", 2019.

[21]H. Wang, D. He and H. Wang, "Comparison of High-Performance Pacl<et Processing Frameworks on NUMA", Beijing Laboratory of Advanced Information Network, Beijing, 2019.

[22]T. Barbette, C. Soldani and L. Mathy, "Fast Userspace Packet Processing", University of Liege, Belguim, 2019.